Packet Route Tracer

Version 1.2

Prepared By

Carnivore Mafia

December 15th, 2015

Table of Contents

List of Figures

List of Tables

No table of figures entries found.

Record of Changes

*A-Added M-Modified D-Deleted

CHANGE NUMBER	DATE	NUMBER OF FIGURE, TABLE OR PARAGRAPH	*A M D	TITLE OR BRIEF DESCRIPTION	CHANGE REQUEST NUMBER
1.0	11/16/2015		A	Initial Submission	
1.1	12/4/2015		M	First Corrections	
1.2	12/15/2015		A	Final Submission	

Project Design Document

---

1 INTRODUCTION

1.1 Application Overview

Packet Route Tracer is a network tool that will allow users to visualize a packet's possible route given a specific domain name to a destination. The application is intended to run on Mac OSX/Linux and will trace the route a packet may take from the host machine to a given domain and display the route on a user-interactive graphical user interface.

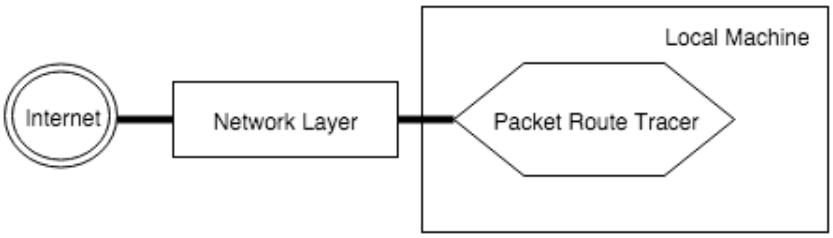

Figure 1: Packet Route Tracer uses the Network Layer to gather route information.

1.2 Definitions, Acronyms, Abbreviations

- **Packet:** a packet is a segment of data sent between two hosts/devices on a network
- **IP Address:** a unique string of integers separated by periods that identifies each computer's location on a network
- **Domain name:** a string that identifies a single IP address or a set of IP addresses
- **ICMP:** a reporting protocol used to send error messages such as "the requested service is not available" between network devices
- **Routes/paths:** a series of devices that a packet must visit to reach the desired destination
- **GUI:** Graphical User Interface.
- **Python:** a programming language with built in network programming libraries

2 TEAM MEMBERS AND RESPONSIBILITES

2.1 Team Members

- Vinicius Romani
- Romeo Bellon
- Jose Gonzalez

2.2 Responsibilities

Vinicius Romani shall be responsible for the following requirements:

- 3.1.1
- 3.1.2
- 3.1.3
- 3.1.4
- 3.1.5
- 3.1.6
- 3.1.7
- 3.1.8
- 3.1.9
- 3.1.10

Romeo Bellon shall be responsible for the following requirements:

- 3.2.1
- 3.2.2
- 3.2.3
- 3.2.4
- 3.2.5

Jose Gonzalez shall be responsible for the following requirements:

- 4.1.1
- 4.1.2
- 4.1.3
- 4.1.4

Optional functional requirements:

- 3.1.10 (Geolocation Mapping) will be implemented by Romeo Bellon and Jose Gonzalez should time allow once all of the functional requirements are met.

3 FUNCTIONAL REQUIREMENTS

3.1 General Requirements

3.1.1 Open and close sockets as needed throughout the algorithm.

3.1.2 Use the ICMP protocol to work with packets to determine routes.

3.1.3 Manipulate the Time to Live (TTL) field.

3.1.4 Detect the network devices along the packet path.

3.1.5 Send packets along the path.

3.1.6 Repeat trace route algorithm until the destination is reached.

3.1.7 Provide an ordered list of network devices encountered along the calculated path.

3.1.8 Provide a method for users to start and terminate the packet tracing algorithm.

3.1.9 Determine the geolocation of a network device along the path.

3.1.10 (Optional): Map the network devices to a Google map using the geolocations of the devices.

3.2 User Interface Requirements

3.2.1 The GUI will be interactive, allowing for the user to input a desired domain name.

3.2.2 The GUI will display all available controls to the user.

3.2.3 The GUI will display a list of the clients traversed by the packet tracing algorithm.

3.2.4 The GUI will display the routes intuitively in a manner that can be understood easily by the user.

3.2.5 The GUI will display all of the network devices encountered and the relevant information for each device.

4 RESOURCES AND MATERIALS

4.1 Non-functional Requirements

4.1.1 The program will use Java and Python together to create a functional and reliable application.

4.1.2 JavaFX will be used to efficiently create a simple and user friendly application.

4.1.3 Python will be used to implement the route-tracing algorithm involved.

4.1.4 Packet Route Tracer will be designed for OSX/Linux and will be launched by clicking on a .jar file.

5 DESIGN OVERVIEW

For the user interface of the tool, which is going to be responsible to define how the route and the paths will be visually shown to the user, we are going to use a Java platform called JavaFX (http://docs.oracle.com/javase/8/javase-clienttechnologies.htm). It is a tool that has been developed for GUI design allowing developers to create, test, debug and deploy applications.

The trace route implementation will be coded using the Python inter-process communication and networking library (https://docs.python.org/2/library/ipc.html). This library has all of the tools needed to complete the trace route algorithm. An overview of the algorithm to find the path to the destination is provided below.

1. User will input the desired destination domain name.
2. The host client will send an ICMP Echo Request packet with a TTL (IP Time to Live) field value of 1. This value is the number of hops/number of seconds taken by the packet.
3. The router that receives this packet must then drop the packet because the TTL has expired.
4. The receiving router will then send an ICMP TTL Expired message back to the host client.
5. The host client will examine the source IP of the message and note it as the first router along the path.
6. The host client will then increment the TTL value (to 2 in this case) and send another set of ICMP Echo Request messages.
7. The starting router on the path will then decrement the TTL (in this case down to 1) and then forward the remaining packets on to the next router along the path.
8. The receiving router will then send an ICMP TTL Expired message back to the host client.
9. This algorithm will be repeated until the destination IP address has been reached.

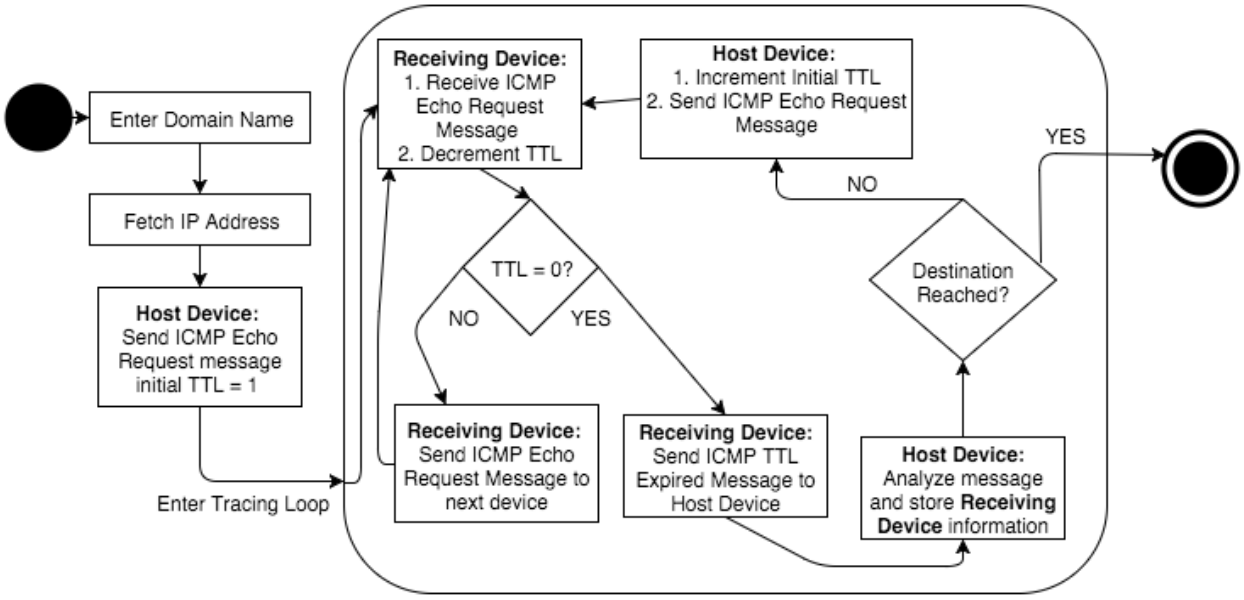

Figure 2: Typical program flow for Packet Route Tracer.

To create the *Packet Route Tracer*, it was necessary to use two languages, Python and Java. As a result, our application has two components: the *Traceroute* class and a JavaFX GUI with a controller to control the *Traceroute* class. Figure 3 below is a visualization of the layered architecture that we used for *Packet Route Tracer*. Because of the JavaFX controller, the GUI and *Traceroute* class communicate directly. The way Packet Route Tracer accomplishes this is by creating a separate thread within the JavaFX controller's bounds that initializes the *Traceroute* class and runs its methods.

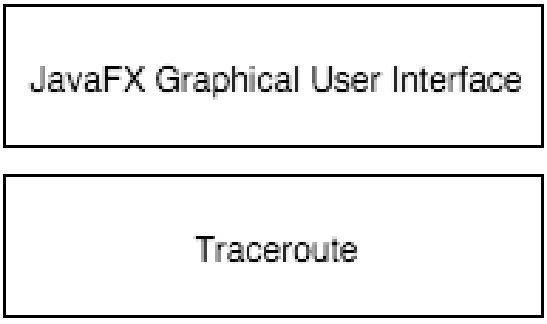

Figure 3: Overview of Packet Route Tracer's layered architecture.

Traceroute will include the required functions and supporting variables to initialize a *Traceroute* object, retrieve IP addresses, open and close sockets, create and send packets and retrieve information from sockets.

JavaFX Controller/GUI will include the required functions and supporting variables to initialize a session, manage event-driven functions, display all of the pertinent information to the user and create a visual representation of a packet's possible route. Additionally it will be responsible for all of the controls and features needed by the user to operate the application.

6 DETAILED DESIGN

6.1 Packet Tracing Component

In order to create this application, we created a Python class *Traceroute* to support an Object-Oriented implementation. This class is responsible for managing all of the pertinent data for the route tracing algorithm and supporting state changes throughout the packet tracing algorithm. The data includes things such as the IP addresses, the domain names, the TTL, among other things. Below is the class diagram for the *Traceroute* class:

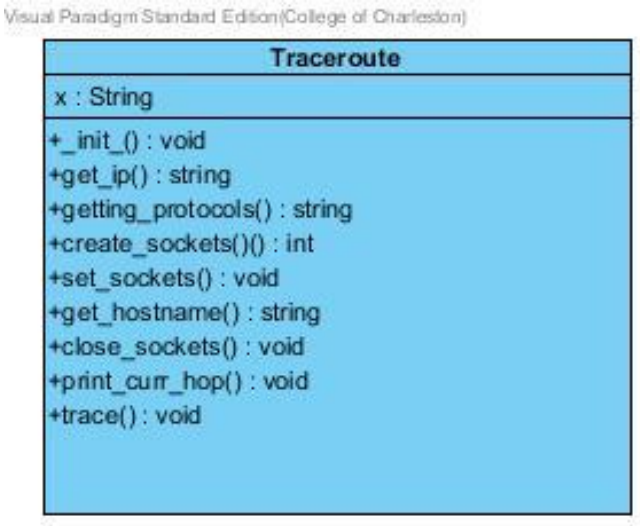

Figure 4: Traceroute class diagram.

There are a few steps and functions required to trace the path of a packet with the Python Networking API. A general overview of these steps and functions are:

1. First, a trace object that will hold all pertinent information in the tracing algorithm must be initialized. The information includes the destination name, port number, maximum hops, TTL,

the current IP address, the current domain name and a list of the network devices encountered along the path.

2. Next, because the application user input is the domain name, the application must fetch the IP address of the given domain. This is accomplished with a getter function get_ip(), with the parameter being the trace object.

3. Now that the IP address is assigned to the trace object, it is possible to fetch the ICMP and UDP and create a packet to be sent.

4. A sending and receiving socket must then be created and set, using the create_sockets() and set_sockets() functions.

5. Once all of the above has been completed, the application sends the packet out and starts a while loop attempting to gather information from the receiving sockets during each pass. If successful, it will take the data along with the IP address and port, and will get the hostname of the device by using the get_hostname() function.

6. Once the information is gathered, the device information will be printed out to a buffer, which then can be read by the JavaFX Controller for use within the GUI.

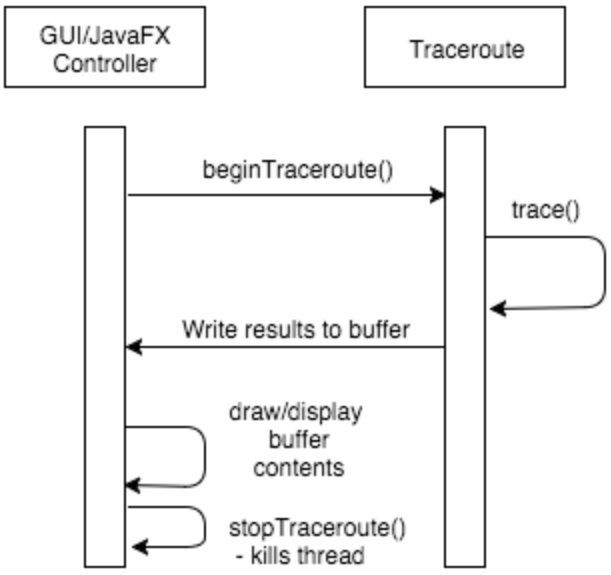

Figure 5: Sequence diagram of interactions between the *GUI/JavaFX Controller* and *Traceroute* class.

6.2 Graphical User Interface (GUI) Component

The graphical user interface methods provided by the JavaFX platform are numerous but for Packet Route Tracer, there were only a few that were needed for the GUI. When the application is opened, the GUI initializes the *Stage,* which is the GUI window. The JavaFX controller class then initializes and places all of the buttons, labels, text fields and any other objects by using an FXML style sheet.

A JavaFX application is based off of three components; a main which initializes a stage (or window), an FXML style sheet which styles and helps control all of the objects in the GUI, and a controller class which holds any functions that need to be accessed by the GUI. The main components of the Packet Route Tracer GUI are a start and stop button for the trace implementation, a loading bar to show that the application is running, a line chart that visually displays the route retrieved, and a list view that shows all of the network devices along the calculated route.

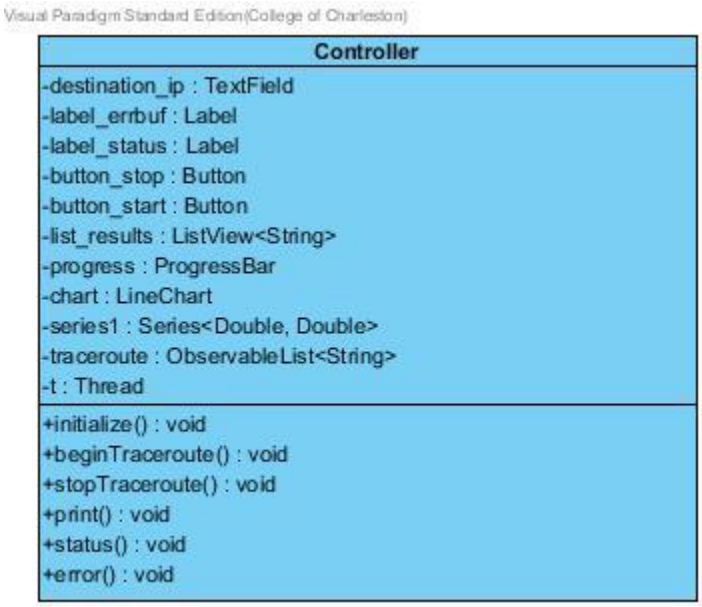

Figure 6: Class diagram for the JavaFX Controller.